# 银行家算法问题

***问题描述：***

**银行家算法**（Banker's Algorithm）是一个避免[死锁](http://zh.wikipedia.org/wiki/%E6%AD%BB%E9%94%81)（Deadlock）的著名算法，是由[艾兹格·迪杰斯特拉](http://zh.wikipedia.org/wiki/%E8%89%BE%E5%85%B9%E6%A0%BC%C2%B7%E8%BF%AA%E6%9D%B0%E6%96%AF%E7%89%B9%E6%8B%89)在1965年为T.H.E系统设计的一种避免死锁产生的算法。它以银行借贷系统的分配策略为基础，判断并保证系统的安全运行。

在银行中，客户申请贷款的数量是有限的，每个客户在第一次申请贷款时要声明完成该项目所需的最大资金量，在满足所有贷款要求时，客户应及时归还。银行家在客户申请的贷款数量不超过自己拥有的最大值时，都应尽量满足客户的需要。在这样的描述中，银行家就好比操作系统，资金就是资源，客户就相当于要申请资源的进程。

***代码描述：***

Available //利用资源向量

Max //最大需求矩阵

Allocation //分配矩阵

Need //每个进程还需要的剩余资源

Need[i][j] = Max[i][j] - Allocation[i][j]

Request[ i ] //表示进程Pi进程的申请向量，如 Request[ i ][ j ]=m表示Pi申请m个Rj类资源

Free[ j ] //表示系统可分配给各进程的Rj类资源数目，初始与当前Available等值

Finish[ i ] //表示进程Pi在此次检查中是否被满足，初始均为false

//当有足够的资源分配给进程时，finish[i]=true,pi完成释放资源(Free[j]+=Allocation[i][j])

if( Request[ i ][ j ]<=Need[ i ][ j ] && Request[ i ][ j ]<=Available[ j ] )

/\*检测资源申请量是否满足说明最大值和可调用的最大值，前者判断不合格会导致程序出错（因为它不符合说明），后者出错会导致程序wait，对后续进程需求进行检测，若出现不符合的则安全检查结束，当前进程进入等待\*/

{

Available[ j ] = Available[ j ] - Request[ i ][ j ];

Allocation[ i ][ j ] = Allocation[ i ][ j ] + Request[ i ][ j ];

Need[ i ][ j ] = Need[ i ][ j ] - Request[ i ][ j ];

/\*可分配的量、进程已经分配到的资源、进程还剩多少资源更新\*/

int trueSum=0, i=0 ;

boolean Flag=true;

while( trueSum<P.length-1 && Flag==true ){

/\*结束的条件是判断完所有的进程均安全或者出现任一不安全的进程\*/

i=i%P.length;

Available[ j ] = Available[ j ] - Request[ i ][ j ];

Allocation[ i ][ j ]=Allocation[ i ][ j ] + Request[ i ][ j ];

Need[ i ][ j ]=Need[ i ][ j ]— Request[ i ][ j ];

//对释放的资源进行重现更新

if( Finish[ i ]==false ){

if(Need[ i ][ j ]<Free[ j ]){

Free[ j ]+ =Allocation[ i ][ j ];

Finish[ i ]=true;

trueSum++;

i++;

}

else

Flag=false;

/\*判断进程是否安全，并且更新free的空间（free即下次可分配的空间）\*/

if( Flag==false)

pass;

else

nopass;

}

}